



ЦЕНТР
ДОПОЛНИТЕЛЬНОГО
ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА по курсу «Data Science»

Докладчик: Матюнин Александр Александрович



Цель и задача работы

Целью работы является изучение теоретических основ и методов решения поставленной задачи. Создать и разработать приложение, которое будет предсказывать ряд конечных свойств композиционных материалов основываясь на предобученных моделях.

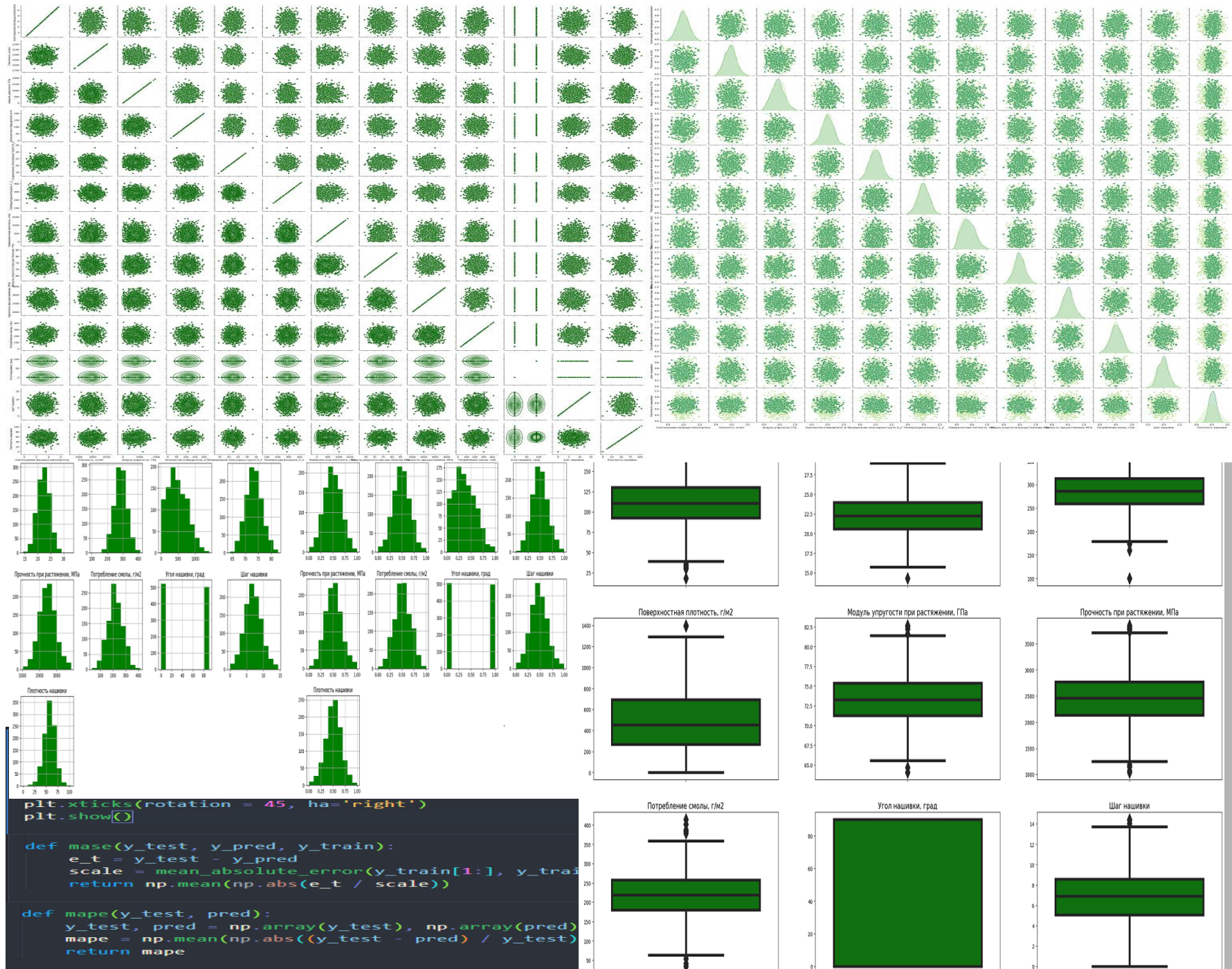
Задачи данной работы:

- Изучить теоретические основы и методы решения поставленной задачи.
- Провести разведочный анализ предложенных данных. Необходимо нарисовать гистограммы распределения каждой из переменной, диаграммы ящика с усами, попарные графики рассеяния точек. Необходимо также для каждой колонке получить среднее, медианное значение, провести анализ и исключение выбросов, проверить наличие пропусков.
- Провести предобработку данных
- Обучить нескольких моделей для прогноза
- Написать нейронную сеть
- Разработать приложение с графическим интерфейсом
- Создать репозиторий в GitHub / GitLab и разместить там код исследования



Начало работы:

- Изучил теоретические основы, методы решения и практические составляющие
- Строил графики
- Старался делать графики в одном стиле
- Создавал понятные переменные, что бы можно было использовать в дальнейшем
- Создавал функции





Объединение файлов и разведочный анализ:

- Импорт необходимых библиотек
- Загрузка файлов
- Проверка размерности
- Объединение файлов по индексу, тип объединения INNER
- Посмотрим на данные объединенного датасета
- Изучим о нем информацию
- Проверим типы данных в каждом столбце
- Проверим на пропуски и дубликаты
- Посмотрим уникальные значения с помощью функции nunique

```
import numpy as np
import pandas as pd
import seaborn
import seaborn as sns
import matplotlib.pyplot as plt
import scipy
from scipy import stats
%matplotlib inline
import sklearn
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input, BatchNormalization, LeakyReLU, Activation, Dropout
import tensorflow as tf
import keras
from tensorflow.keras import layers
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam

print(f'numpy ver: {np.__version__}')
print(f'pandas ver: {pd.__version__}')
print(f'seaborn ver: {sns.__version__}')
print(f'sklearn ver: {sklearn.__version__}')
print(f'scipy ver: {scipy.__version__}')
print(f'tensorflow ver: {tensorflow.__version__}')
print(f'keras ver: {keras.__version__}')

numpy ver: 1.24.3
pandas ver: 2.0.3
seaborn ver: 0.12.2
sklearn ver: 1.3.0
scipy ver: 1.11.2
tensorflow ver: 2.13.0
keras ver: 2.13.1
```

```
# первую колонку (порядковый номер) используем как индекс
df_bp = pd.read_excel(r'C:\Users\Александр\Desktop\VKR_KOMPOZIT\Data\X_bp.xlsx', index_col = 'Unnamed: 0')
df_nup = pd.read_excel(r'C:\Users\Александр\Desktop\VKR_KOMPOZIT\Data\X_nup.xlsx', index_col = 'Unnamed: 0')
```

```
# смотрим количество строк в датасете(размерность строк)
print(f'Количество строк в датасете X_bp.xlsx {df_bp.shape[0]}')
print(f'Количество строк в датасете X_nup.xlsx {df_nup.shape[0]}')
```

Количество строк в датасете X_bp.xlsx 1023
Количество строк в датасете X_nup.xlsx 1040

```
# смотрим количество колонок в датасете(размерность колонок)
print(f'Количество колонок в датасете X_bp.xlsx {df_bp.shape[1]}')
print(f'Количество колонок в датасете X_nup.xlsx {df_nup.shape[1]}')
```

Количество колонок в датасете X_bp.xlsx 10
Количество колонок в датасете X_nup.xlsx 3

```
# Эти два датасета имеют разный объем строк.
# Наша задача собрать исходные данные в один, единый набор данных.
# В pandas есть метод объединения по индексу INNER
df_nb = pd.merge(df_bp, df_nup, how="inner", left_on=None, right_index=True, right_index=True)
df_nb.head()
```

	Соотношение матрица-наполнителя	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп, %	Температура вспыхивания, C.2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2	Угол нашивки, град	Шаг нашивки	Плотность нашивки
0	1.857143	2030.0	738.736842	30.00	22.267857	100.000000	210.0	70.0	3000.0	220.0	0	4.0	57.0
1	1.857143	2030.0	738.736842	50.00	23.750000	284.615385	210.0	70.0	3000.0	220.0	0	4.0	60.0
2	1.857143	2030.0	738.736842	49.90	33.000000	284.615385	210.0	70.0	3000.0	220.0	0	4.0	70.0
3	1.857143	2030.0	738.736842	129.00	21.250000	300.000000	210.0	70.0	3000.0	220.0	0	5.0	47.0
4	2.771331	2030.0	753.000000	111.86	22.267857	284.615385	210.0	70.0	3000.0	220.0	0	5.0	57.0

```
# смотрим размерность объединенного датафрейма
df_nb.shape
```

Итоговый датасет имеет 13 столбцов и 1023 строки, 17 строк из таблицы df_nup было отброшено, т.е. часть данных была удалена на начальном этапе исследования.
При использовании "Inner", из результатов запроса будут удалены все строки, которые не нашлись соответствующая пара в другой таблице.

```
df_nb.info()
```

```
df_nb.describe()
```

```
df_nb.nunique()
```

```
df_nb.value_counts()
```

```
df_nb.isnull().sum()
```

```
df_nb.dtypes
```

```
df_nb.head()
```

```
df_nb.tail()
```

```
df_nb.sample(5)
```

```
df_nb.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Соотношение матрица-наполнителя	1023.0	2.930366	0.913322	0.300000	2.317287	2.900000	3.500000	5.000000
Плотность, кг/м3	1023.0	1978.734888	73.729231	1731.764635	1924.155467	1977.621657	2021.374378	2207.724481
модуль упругости, ГПа	1023.0	738.736842	330.231581	243.666667	500.000000	738.736842	961.012162	1911.536872
Количество отвердителя, м.%	1023.0	110.570769	36.295911	17.740275	92.443497	110.564840	129.730366	198.953207
Содержание эпоксидных групп, %	1023.0	22.244390	2.406301	14.250000	20.000000	22.230744	23.961934	33.000000
Температура вспыхивания, C.2	1023.0	285.863151	40.943260	100.000000	255.056612	285.056612	313.000100	413.273418
Поверхностная плотность, г/м2	1023.0	482.731833	281.314690	0.000000	266.816645	451.864365	693.220177	1399.542362
Модуль упругости при растяжении, ГПа	1023.0	73.320571	3.110063	64.000000	71.245018	73.245018	75.366012	82.682051
Прочность при растяжении, МПа	1023.0	2466.922843	485.628000	1036.856605	2135.850448	2459.524526	2767.193119	3848.436732
Потребление смолы, г/м2	1023.0	218.231448	59.729231	33.000000	179.627520	219.198882	257.481724	414.000000
Угол нашивки, град	1023.0	44.252199	45.015793	0.000000	0.000000	0.000000	0.000000	90.000000
Шаг нашивки	1023.0	6.899222	2.563467	0.000000	5.000000	6.916144	8.586293	14.440222
Плотность нашивки	1023.0	57.153929	12.350000	0.000000	49.749112	57.341920	64.944861	103.988001

```
df_nb.dtypes
```

	dtype
Соотношение матрица-наполнителя	float64
Плотность, кг/м3	float64
модуль упругости, ГПа	float64
Количество отвердителя, м.%	float64
Содержание эпоксидных групп, %	float64
Температура вспыхивания, C.2	float64
Поверхностная плотность, г/м2	float64
Модуль упругости при растяжении, ГПа	float64
Прочность при растяжении, МПа	float64
Потребление смолы, г/м2	float64
Угол нашивки, град	float64
Шаг нашивки	float64
Плотность нашивки	float64

```
df_nb.head()
```

	Соотношение матрица-наполнителя	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп, %	Температура вспыхивания, C.2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2	Угол нашивки, град	Шаг нашивки	Плотность нашивки
0	1.857143	2030.0	738.736842	30.00	22.267857	100.000000	210.0	70.0	3000.0	220.0	0	4.0	57.0
1	1.857143	2030.0	738.736842	50.00	23.750000	284.615385	210.0	70.0	3000.0	220.0	0	4.0	60.0
2	1.857143	2030.0	738.736842	49.90	33.000000	284.615385	210.0	70.0	3000.0	220.0	0	4.0	70.0
3	1.857143	2030.0	738.736842	129.00	21.250000	300.000000	210.0	70.0	3000.0	220.0	0	5.0	47.0
4	2.771331	2030.0	753.000000	111.86	22.267857	284.615385	210.0	70.0	3000.0	220.0	0	5.0	57.0



Визуализация «сырых» данных:

- Построение гистограмм распределения
- Диаграмм Box Plot
- Попарные графики рассеяния точек
- Тепловые карты

Диаграммы "ящики с усами" для определения выбросов

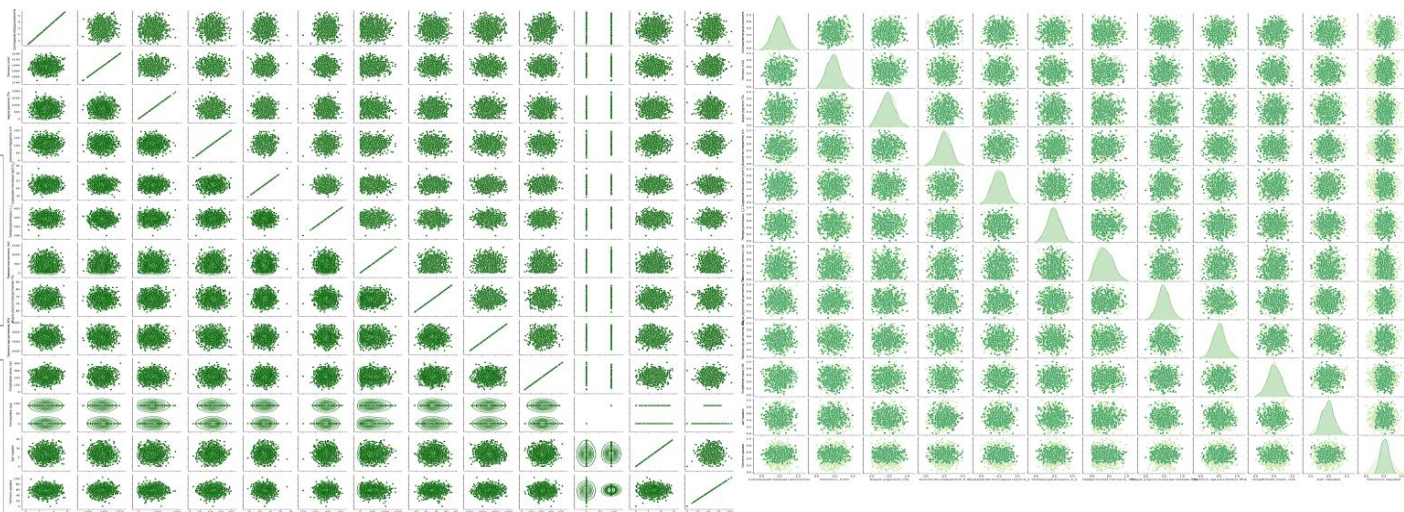
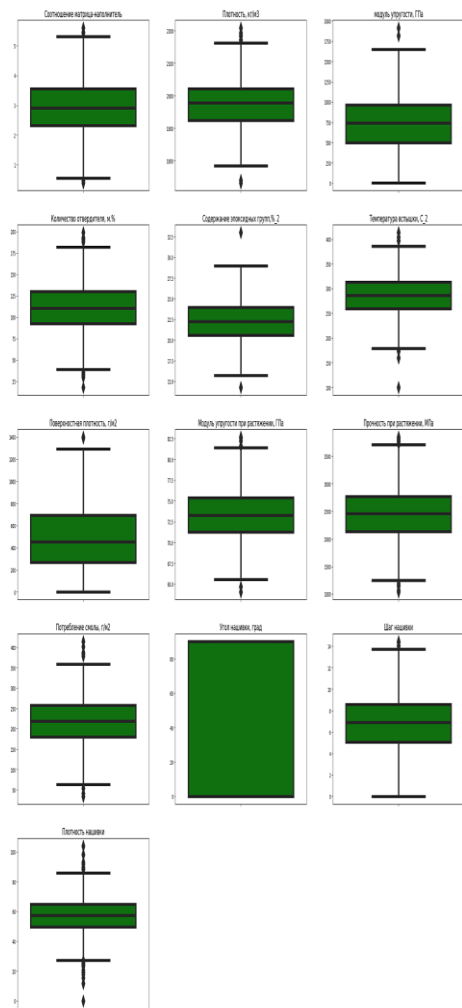
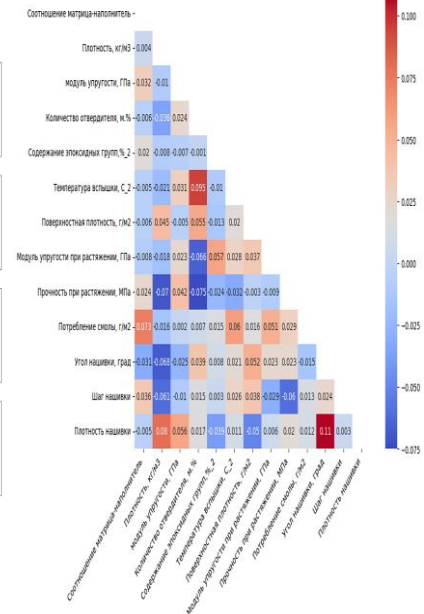
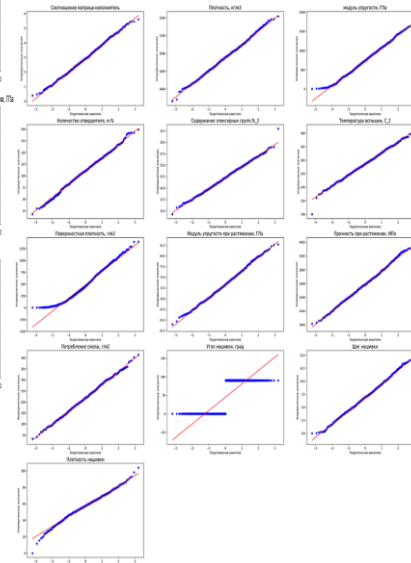
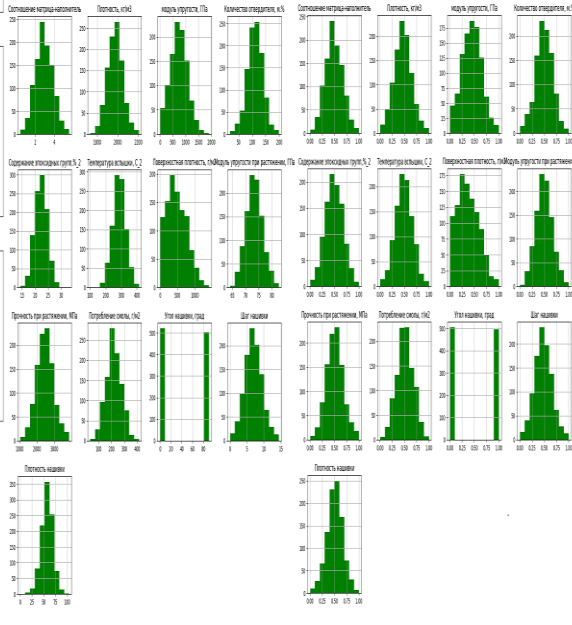


График вероятности для распределения данных





Предобработка данных:

- Подсчет количества значений методом 3 сигм и методом межквартильных расстояний
- Исклучим выбросы методом 3 сигм
- Посмотрим описание

```
# методом 3-х сигм
count_3s = 0
for column in df_nb:
    d = df_nb.loc[:, [column]]
    zscore = (df_nb[column] - df_nb[column].mean()) / df_nb[column].std()
    d['3s'] = zscore.abs() > 3
    count_3s += d['3s'].sum()

print('Метод 3-х сигм, выбросов:', count_3s)
```

Метод 3-х сигм, выбросов: 24

```
# метод межквартильных расстояний
count_iq = 0
for column in df_nb:
    d = df_nb.loc[:, [column]]
    q1 = np.quantile(df_nb[column], 0.25)
    q3 = np.quantile(df_nb[column], 0.75)
    iqr = q3 - q1
    lower = q1 - 1.5 * iqr
    upper = q3 + 1.5 * iqr
    d['iq'] = (df_nb[column] <= lower) | (df_nb[column] >= upper)
    count_iq += d['iq'].sum()

print('Метод межквартильных расстояний, выбросов:', count_iq)
```

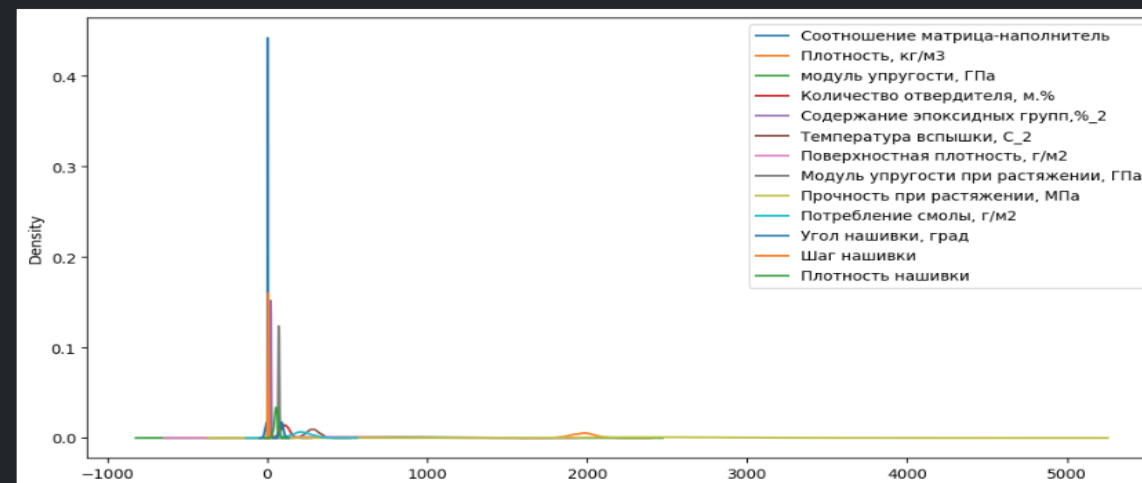
Метод межквартильных расстояний, выбросов: 93

```
# Удаляем выбросы методом 3-х сигм
outliers = pd.DataFrame(index=df_nb.index)
for column in df_nb:
    zscore = (df_nb[column] - df_nb[column].mean()) / df_nb[column].std()
    outliers[column] = (zscore.abs() > 3)
df_nb_clear = df_nb[outliers.sum(axis=1)==0]
df_nb_clear.shape
```

(1000, 13)

```
fig, ax = plt.subplots(figsize = (12, 6))
df_nb_clear.plot(kind = 'kde', ax = ax)
```

<Axes: ylabel='Density'>



```
minmax_and_mean50_cl = df_nb_clear.describe()
minmax_and_mean50_cl.loc[['min', 'max', 'mean', '50%']].T
```

	min	max	mean	50%
Соотношение матрица-наполнитель	0.389403	5.591742	2.936299	2.908811
Плотность, кг/м3	1784.482245	2192.738783	1975.402478	1977.321002
модуль упругости, ГПа	2.436909	1649.415706	738.675486	741.148111
Количество отвердителя, м.%	29.956150	192.851702	110.821904	110.652620
Содержание эпоксидных групп, %	15.695894	28.955094	22.235549	22.221462
Температура вспышки, C_2	173.484920	403.652861	285.957299	285.853960
Поверхностная плотность, г/м2	0.603740	1291.340115	479.855825	450.869535
Модуль упругости при растяжении, ГПа	64.054061	82.682051	73.318178	73.230375
Прочность при растяжении, МПа	1036.856605	3848.436732	2464.864198	2456.394188
Потребление смолы, г/м2	41.048278	386.903431	218.254011	218.697660
Угол нашивки, град	0.000000	90.000000	44.640000	0.000000
Шаг нашивки	0.037639	14.440522	6.910600	6.922196
Плотность нашивки	20.571633	92.963492	57.276293	57.471971



Предобработка данных:

- Нормализуем данные MinMaxScaler()
- Построим график плотности
- Посмотрим описание
- Выведем описательную статистику

приводим данные к масштабированию с помощью MinMaxScaler

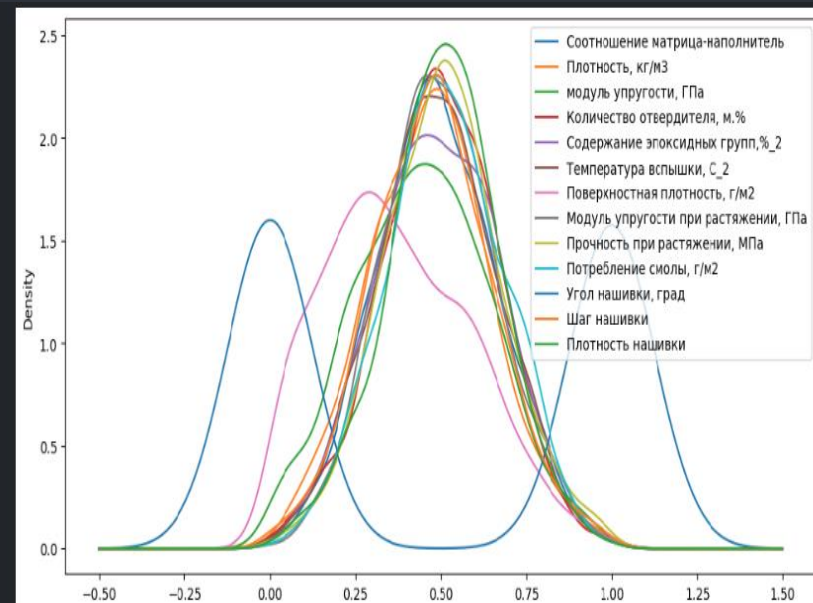
```
scaler = MinMaxScaler()  
df_min_max = pd.DataFrame(scaler.fit_transform(df_nb_clear), columns=column_names)
```

```
df_min_max.head()
```

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп, %_2	Температура вспышки, С_2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2	Угол нашивки, град	Шаг нашивки	Плотность нашивки
0	0.282131	0.601381	0.447061	0.123047	0.607435	0.482823	0.16223	0.319194	0.698235	0.517418	0.0	0.275109	0.544652
1	0.282131	0.601381	0.447061	0.608021	0.418887	0.549664	0.16223	0.319194	0.698235	0.517418	0.0	0.344539	0.365074
2	0.457857	0.601381	0.455721	0.502800	0.495653	0.482823	0.16223	0.319194	0.698235	0.517418	0.0	0.344539	0.503211
3	0.457201	0.527898	0.452685	0.502800	0.495653	0.482823	0.16223	0.319194	0.698235	0.517418	0.0	0.344539	0.544652
4	0.419084	0.307448	0.488508	0.502800	0.495653	0.482823	0.16223	0.319194	0.698235	0.517418	0.0	0.344539	0.682789

```
df_min_max.describe().T
```

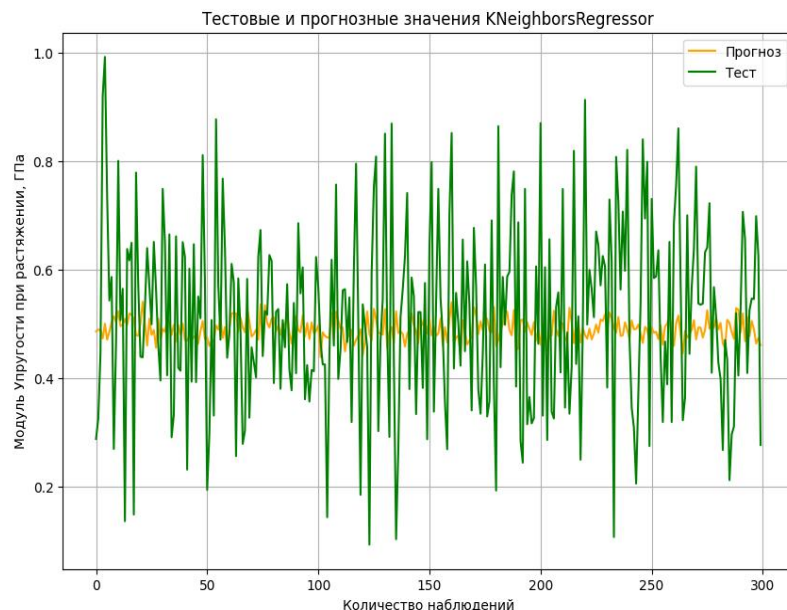
	count	mean	std	min	25%	50%	75%	max
Соотношение матрица-наполнитель	1000.0	0.489568	0.174687	0.0	0.370964	0.484284	0.608289	1.0
Плотность, кг/м3	1000.0	0.467648	0.178696	0.0	0.340831	0.472347	0.579727	1.0
модуль упругости, ГПа	1000.0	0.447024	0.198876	0.0	0.302576	0.448525	0.582408	1.0
Количество отвердителя, м.%	1000.0	0.496427	0.171089	0.0	0.384097	0.495388	0.613258	1.0
Содержание эпоксидных групп, %_2	1000.0	0.493216	0.179818	0.0	0.368597	0.492154	0.624396	1.0
Температура вспышки, С_2	1000.0	0.488654	0.174792	0.0	0.371985	0.488205	0.606271	1.0
Поверхностная плотность, г/м2	1000.0	0.371301	0.215155	0.0	0.206374	0.348844	0.535295	1.0
Модуль упругости при растяжении, ГПа	1000.0	0.497322	0.167158	0.0	0.386234	0.492609	0.605138	1.0
Прочность при растяжении, МПа	1000.0	0.507902	0.172506	0.0	0.390414	0.504890	0.612932	1.0
Потребление смолы, г/м2	1000.0	0.512370	0.170432	0.0	0.401220	0.513653	0.625772	1.0
Угол нашивки, град	1000.0	0.496000	0.500234	0.0	0.000000	0.000000	1.000000	1.0
Шаг нашивки	1000.0	0.477193	0.177586	0.0	0.351886	0.477999	0.593714	1.0
Плотность нашивки	1000.0	0.507027	0.163634	0.0	0.405037	0.509730	0.612766	1.0





Разработка и обучение моделей для прогноза упругости при растяжении, ГПа

- Построение моделей
- Обучение моделей
- Вычисление ошибок
- Построение графиков
- Поиск гиперпараметров в методе GridSearchCV с перекрестной проверкой с количеством блоков 10
- Подставление оптимальных параметров



```
X_uprygost = df_min_max.drop(["Модуль упругости при растяжении, ГПа"], axis=1)
X_prochnost = df_min_max.drop(["Прочность при растяжении, МПа"], axis=1)
y_uprygost = df_min_max["Модуль упругости при растяжении, ГПа"]
y_prochnost = df_min_max["Прочность при растяжении, МПа"]

X_uprygost_train, X_uprygost_test, y_uprygost_train, y_uprygost_test = train_test_split(X_uprygost, y_uprygost, test_size = 0.3, shuffle = True)
X_prochnost_train, X_prochnost_test, y_prochnost_train, y_prochnost_test = train_test_split(X_prochnost, y_prochnost, test_size = 0.3, shuffle = True)

print('X_train: для Модуль упругости при растяжении, ГПа', X_uprygost_train.shape, 'y_train: для Модуль упругости при растяжении, ГПа', y_uprygost_train.shape)
print('X_test: для Модуль упругости при растяжении, ГПа', X_uprygost_test.shape, 'y_test: для Модуль упругости при растяжении, ГПа', y_uprygost_test.shape)
print('X_train: для Прочность при растяжении, МПа, ГПа', X_prochnost_train.shape, 'y_train: для Прочность при растяжении, МПа', y_prochnost_train.shape)
print('X_test: для Прочность при растяжении, МПа', X_prochnost_test.shape, 'y_test: для Прочность при растяжении, МПа', y_prochnost_test.shape)

X_train: для Модуль упругости при растяжении, ГПа (700, 12) y_train: для Модуль упругости при растяжении, ГПа (700,)
X_test: для Модуль упругости при растяжении, ГПа (300, 12) y_test: для Модуль упругости при растяжении, ГПа (300,)
X_train: для Прочность при растяжении, МПа, ГПа (700, 12) y_train: для Прочность при растяжении, МПа (700,)
X_test: для Прочность при растяжении, МПа (300, 12) y_test: для Прочность при растяжении, МПа (300,)

knn_uprygost = KNeighborsRegressor()
param_grid = {
    "n_neighbors": range(1, 301, 2),
    "weights": ['uniform', 'distance'],
    "algorithm": ['auto', 'ball_tree', 'kd_tree', 'brute'],
    "p": range(1, 5, 2)
}

grid_search_knn_uprygost = GridSearchCV(knn_uprygost, param_grid, cv=10, verbose=1, scoring = 'neg_mean_squared_error')
grid_search_knn_uprygost.fit(X_uprygost_train, y_uprygost_train)
model_knn_uprygost = grid_search_knn_uprygost.best_estimator_
model_knn_uprygost.fit(X_uprygost_train, y_uprygost_train)
knn_uprygost_pred = model_knn_uprygost.predict(X_uprygost_test)
knn_uprygost_Bscore = grid_search_knn_uprygost.best_score_
print("Best params", grid_search_knn_uprygost.best_params_)
print("Best score", knn_uprygost_Bscore)

Fitting 10 folds for each of 2400 candidates, totalling 24000 fits
Best params {'algorithm': 'auto', 'n_neighbors': 73, 'p': 3, 'weights': 'uniform'}
Best score -0.02714483286176299
```

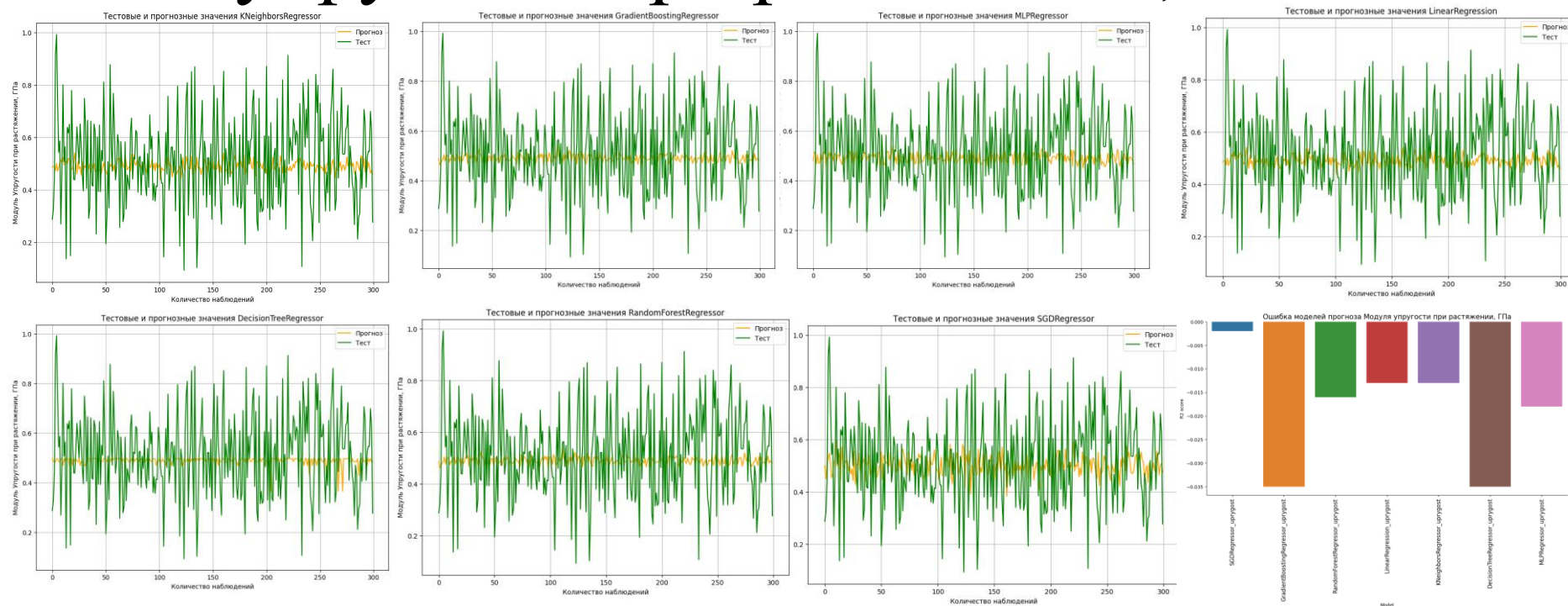
models_uprygost.T							
	Модуль Упругости при растяжении	Модуль Упругости при растяжении	Модуль Упругости при растяжении	Модуль Упругости при растяжении	Модуль Упругости при растяжении	Модуль Упругости при растяжении	Модуль Упругости при растяжении
Model	KNeighborsRegressor_uprygost	LinearRegression_uprygost	SGDRegressor_uprygost	RandomForestRegressor_uprygost	GradientBoostingRegressor_uprygost	MLPRegressor_uprygost	DecisionTreeRegressor_uprygost
MASE	0.744	0.744	0.749	0.747	0.754	0.746	0.755
MAPE	0.339	0.339	0.349	0.34	0.34	0.34	0.343
MSE	0.029	0.029	0.029	0.029	0.03	0.029	0.03
R2 score	-0.013	-0.013	-0.002	-0.016	-0.035	-0.018	-0.035
Best score	-0.027145	-0.039372	-0.076482	-0.040744	-0.062736	-0.020307	-0.025649

models_prochnost.T							
	Модуль Прочность при растяжении	Модуль Прочность при растяжении	Модуль Прочность при растяжении	Модуль Прочность при растяжении	Модуль Прочности при растяжении	Модуль Прочности при растяжении	Модуль Прочности при растяжении
Model	KNeighborsRegressor_prochnost	LinearRegression_prochnost	SGDRegressor_prochnost	RandomForestRegressor_prochnost	GradientBoostingRegressor_prochnost	MLPRegressor_prochnost	DecisionTreeRegressor_prochnost
MASE	0.689	0.697	0.709	0.692	0.733	0.69	0.701
MAPE	inf	inf	inf	inf	inf	inf	inf
MSE	0.03	0.03	0.03	0.03	0.032	0.03	0.03
R2 score	-0.008	-0.006	-0.037	-0.006	-0.083	-0.013	-0.031
Best score	-0.029803	-0.025615	-0.075167	-0.009232	-0.032024	-0.006573	-0.01405



Разработка и обучение моделей для прогноза упругости при растяжении, ГПа

- Графики тестовых и прогнозных значений для разных методов
- График ошибок





Нейронная сеть для соотношения «матрица-наполнитель»:

- Сформируем входы и выходы для модели
- Разобьем на обучающую и тестовую выборки
- Построим модель и посмотрим на результаты
- Обучим нейросеть
- Сохраним модель

```
НС с приведенным к MinMax

df_nn = df_min_max[['Соотношение матрица-наполнитель'], axis=1]
df_nn_nn = df_min_max[['Соотношение матрица-наполнитель'], axis=1]
X_train, X_test, y_train, y_test = train_test_split(df_nn_nn, df_nn,
                                                    test_size = 0.3,
                                                    random_state = 23,
                                                    shuffle = True)

early_nn = EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=1, mode='min')

model_minmax = Sequential()
model_minmax.add(Dense(X_train.shape[1]))
model_minmax.add(Dense(128))
model_minmax.add(Dense(128, activation='relu'))
model_minmax.add(Dense(128, activation='elu'))
model_minmax.add(Dense(128, kernel_regularizer=keras.regularizers.L2(0.01),
                                                    activation='relu'))
model_minmax.add(Dense(24, activation='elu'))
model_minmax.add(Dense(12, activation='relu'))
model_minmax.add(Dense(1, activation='linear'))
```

```
##Построение НС на очищенном наборе данных без нормализации

df_nn_clear = df_nn_clear[['Соотношение матрица-наполнитель'], axis=1]
df_nn_nn_clear = df_nn_clear[['Соотношение матрица-наполнитель'], axis=1]
X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(df_nn_nn_clear, df_nn_clear,
                                                            test_size = 0.3,
                                                            random_state = 23,
                                                            shuffle = True)

normalizer = tf.keras.layers.Normalization(axis=1)

X_train_c_norm = normalizer.adapt(np.array(X_train_c))

model_2 = Sequential(X_train_c_norm)
model_2.add(Dense(X_train_c_norm.shape[1]))
model_2.add(Dense(128, activation='relu'))
model_2.add(Dense(64, kernel_regularizer=keras.regularizers.L2(0.01),
                                                    activation='linear'))
model_2.add(Dense(12, activation='relu'))
model_2.add(Dense(16, activation='relu'))
model_2.add(Dense(1, activation='linear'))

model_2.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3, decay=1e-3 / 200),
                loss='mean_squared_error')
```

```
##Модель НС на не обработанных наборах данных

df = df_nn[['Соотношение матрица-наполнитель'], axis=1]
df_nn_nn_a = df_nn[['Соотношение матрица-наполнитель'], axis=1]
X_train_a, X_test_a, y_train_a, y_test_a = train_test_split(df_nn_nn_a, df,
                                                            test_size = 0.3,
                                                            random_state = 23,
                                                            shuffle = True)

X_train_nn_norm = normalizer.adapt(np.array(X_train_a))

model_nn = Sequential(X_train_nn_norm)
model_nn.add(Dense(128, activation='relu', input_shape=(X_train_a.shape[1],)))
model_nn.add(BatchNormalization())
model_nn.add(LeakyReLU())
model_nn.add(Dense(128, activation='selu'))
model_nn.add(Dropout(0.8))
model_nn.add(Dense(64, activation='selu'))
model_nn.add(Dropout(0.8))
model_nn.add(Dense(12, activation='selu'))
model_nn.add(Dropout(0.8))
model_nn.add(LeakyReLU())
model_nn.add(Dense(16, activation='selu'))
model_nn.add(BatchNormalization())
model_nn.add(Dense(1))
model_nn.add(Activation('selu'))
```

```
model_minmax.summary()

Model: "sequential"
Layer (type) Output Shape Param #
-----
dense_1 (Dense) (None, 128) 1664
dense_2 (Dense) (None, 128) 16512
dense_3 (Dense) (None, 128) 16512
dense_4 (Dense) (None, 24) 3096
dense_5 (Dense) (None, 12) 388
dense_6 (Dense) (None, 1) 13
-----
Total params: 34609 (213.12 KB)
Trainable params: 34609 (213.12 KB)
Non-trainable params: 0 (0.00 Byte)

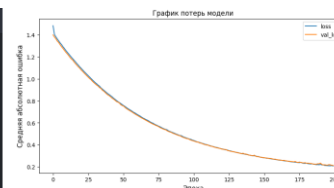
model_minmax.compile(optimizer=tf.optimizers.Adam(learning_rate=0.02, momentum=0.5),
                    loss='mean_absolute_error')
```

```
model_2.summary()

Model: "sequential_1"
Layer (type) Output Shape Param #
-----
dense_7 (Dense) (None, 128) 1664
dense_8 (Dense) (None, 64) 8256
dense_9 (Dense) (None, 32) 2880
dense_10 (Dense) (None, 16) 528
dense_11 (Dense) (None, 1) 17
-----
Total params: 12545 (49.00 KB)
Trainable params: 12545 (49.00 KB)
Non-trainable params: 0 (0.00 Byte)
```

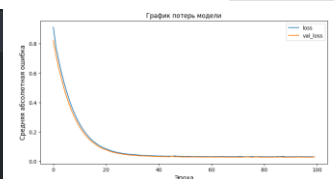
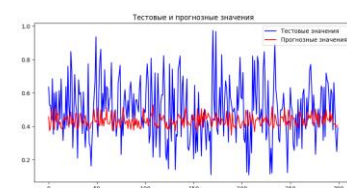
```
model_nn.summary()

leaky_re_lu_3 (LeakyReLU) (None, 128) 0
dense_21 (Dense) (None, 128) 16512
dropout_1 (Dropout) (None, 128) 0
dense_22 (Dense) (None, 64) 8256
dropout_2 (Dropout) (None, 64) 0
dense_23 (Dense) (None, 32) 2080
dropout_3 (Dropout) (None, 32) 0
leaky_re_lu_4 (LeakyReLU) (None, 32) 0
dense_24 (Dense) (None, 16) 528
batch_normalization_7 (BatchNormalization) (None, 16) 64
dense_25 (Dense) (None, 1) 17
activation_1 (Activation) (None, 1) 0
-----
Total params: 29633 (115.75 KB)
Trainable params: 29345 (114.63 KB)
Non-trainable params: 288 (1.12 KB)
```



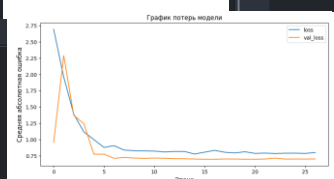
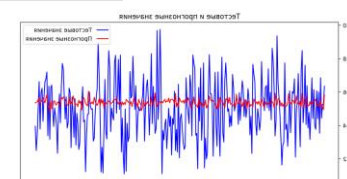
```
model_minmax.save('saved_model/model_minmax.h5')

C:\Users\Александр\AppData\Roaming\Python\Python311\si
instead the native Keras format, e.g. `model.save('my_
saving_api.save_model()
```



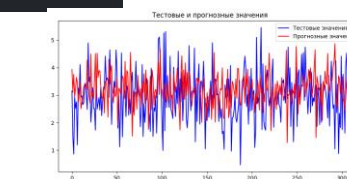
```
model_2.save('saved_model/model_clear.h5')

C:\Users\Александр\AppData\Roaming\Python\Pyt
instead the native Keras format, e.g. `model.
saving_api.save_model()
```



```
model_nn.save('saved_model/model_not_clear.h5')

C:\Users\Александр\AppData\Roaming\Python\Python311
instead the native Keras format, e.g. `model.save('
saving_api.save_model()
```





Приложение:

- Сохранил модель для разработки веб-приложения для прогнозирования соотношения «модуль упругости при растяжении» в фреймворке Flask
- При запуске приложения, пользователь переходит на: <http://127.0.0.1:5000>
- В открывшемся окне пользователю необходимо ввести в соответствующие ячейки требуемые значения и нажать на кнопку «Рассчитать».
- На выходе пользователь получает результат прогноза для значения параметра «Соотношение «матрица – наполнитель»».
- Приложение успешно
- Репозиторий на github.com
- https://github.com/Matiunin1982/VKR_Kompozit_Matiunin_AA/tree/main

Расчет соотношения матрица-наполнитель

Введите параметры

Введите Плотность, кг/м³

Введите Модуль упругости, ГПа

Введите Количество отвердителя, м.%

Введите Содержание эпоксидных групп, %

Введите Температура вспышки, С_2

Введите Поверхностная плотность, г/м²

Введите Модуль упругости при растяжении, ГПа

Введите Прочность при растяжении, МПа

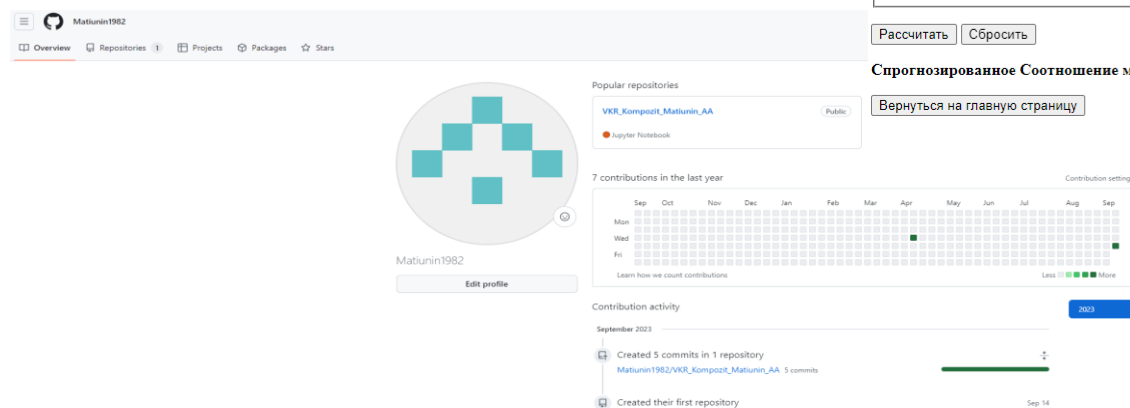
Введите Потребление смолы, г/м²

Введите Угол нашивки, град

Введите Шаг нашивки

Введите Плотность нашивки

Спрогнозированное Соотношение матрица-наполнитель для введенных параметров: [[14.564138]]





Спасибо за
внимание

Трудности и ошибки

- Опечатки, описки, пропуски скобок.
- Составление функций
- Когда что то не получалось или ломалась часть кода приходилось подолгу просматривать различные источники и много раз пробывать
- Самой большой трудностью было выкладывание на гитхаб, часть кода писал в гугл колабе, но приходилось писать и в VSCode, вот с ним то и пришлось помучиться когда нужно было выложить всю работу.
- Пробывал работать в PyCharm, но толи из за того что уже привык к VSCode писалось намного медленнее, хотя все приходит с опытом.



Спасибо за
внимание

Заключение

- Использованные при разработке моделей подходы не позволили получить сколько-нибудь достоверных прогнозов
- Примененные модели регрессии не показали высокой эффективности в прогнозировании свойств композитов
- Невозможно определить из свойств материалов соотношение «матрица – наполнитель»
- Текущим набором алгоритмов текущая задача эффективно не решается.



ЦЕНТР
ДОПОЛНИТЕЛЬНОГО
ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана



do.bmstu.ru